

Prototype Report

< LEMA Pilot School Integrated Family Accountability System >



PROJECT TITLE

LEMA FAMILY ACCOUNTABILITY SYSTEM

TEAM NO

#04

TEAM MEMBERS & ROLES

NAME	ROLES
Teawon Han	Project Manager
Zhen Huang	Feasibility Analyst
Ziming Wei	Operational Concept Engineer
Xiaoli Ma	Life Cycle Planner
Ian Williams	Requirements Engineer
Kimberly Krause	IIV&V / System Requirements Engineer
Ying Yang	Life Cycle Planner

<12/4/2011>

Version History

Date	Author	Version	Changes made	Rationale
09/28/11	IW	1.1	<ul style="list-style-type: none">• First Version	<ul style="list-style-type: none">• N/A
10/02/11	IW	1.2	<ul style="list-style-type: none">• Added elements from Easy Grade Pro• Added student views of reports	<ul style="list-style-type: none">• Client wants product to mimic the look and feel of Easy Grade Pro
10/03/11	Teawon	1.3	<ul style="list-style-type: none">• Change format of document• Update 2.Navigation flow	<ul style="list-style-type: none">• Student could access booklist by web
10/07/11	IW	2.1	<ul style="list-style-type: none">• Format for Core FC Turnin• Add Use Cases	<ul style="list-style-type: none">• Entered Valuation Phase
10/14/11	IW	2.5	<ul style="list-style-type: none">• Update Navigation Flow, Status, and Prototype models after team discussion.	<ul style="list-style-type: none">• TA comments and team discussion found flaws.
10/23/11	IW	2.6	<ul style="list-style-type: none">• Make GUI look more like Easy Grade Pro.• Update use cases to reflect these changes.	<ul style="list-style-type: none">• Next version of FC package due.
11/20/11	IW	3.1	<ul style="list-style-type: none">• Integrate feedback from IIV&V and TA review. Add new prototypes for text messaging and parsing of Easy Grade Pro-exported text files containing student grades.	<ul style="list-style-type: none">• Draft version of DC package due.
12/4/11	IW	3.2	<ul style="list-style-type: none">• Integrate feedback from IIV&V• Add new prototype and use case for the exporting to XML feature.	<ul style="list-style-type: none">• Final version of the DC package.

Table of Contents

Prototype Report	i
Version History	ii
Table of Contents.....	iii
1. Introduction	1
1.1 Purpose of the prototype report	1
1.2 Status of the prototype	1
2. Navigation Flow	2
3. Critical Issues and Risks	3
4. Prototypes.....	4
4.1 UI Look and Feel	4
4.2 Report Content.....	14
4.3 Cross-Project Cooperation.....	14
4.4 Cell Phone Communication Prototype.....	14
4.5 Uploading Data Exported from Easy Grade Pro Prototype	16
5. Use Cases	24
5.1 Simple Use Case	24
5.2 Extended Case with Walkthrough	24
5.3 Uploading Student Grades Use Case.....	26
5.4 Uploading Assignments Use Case	26

1. Introduction

1.1 Purpose of the prototype report

The prototype report is used to mitigate the riskiest aspects of the project. Once all of these aspects are gathered, the prototyper creates prototypes to model proposed solutions for that risk. The client reviews these models and provides feedback about how to improve. This document records the steps in this process.

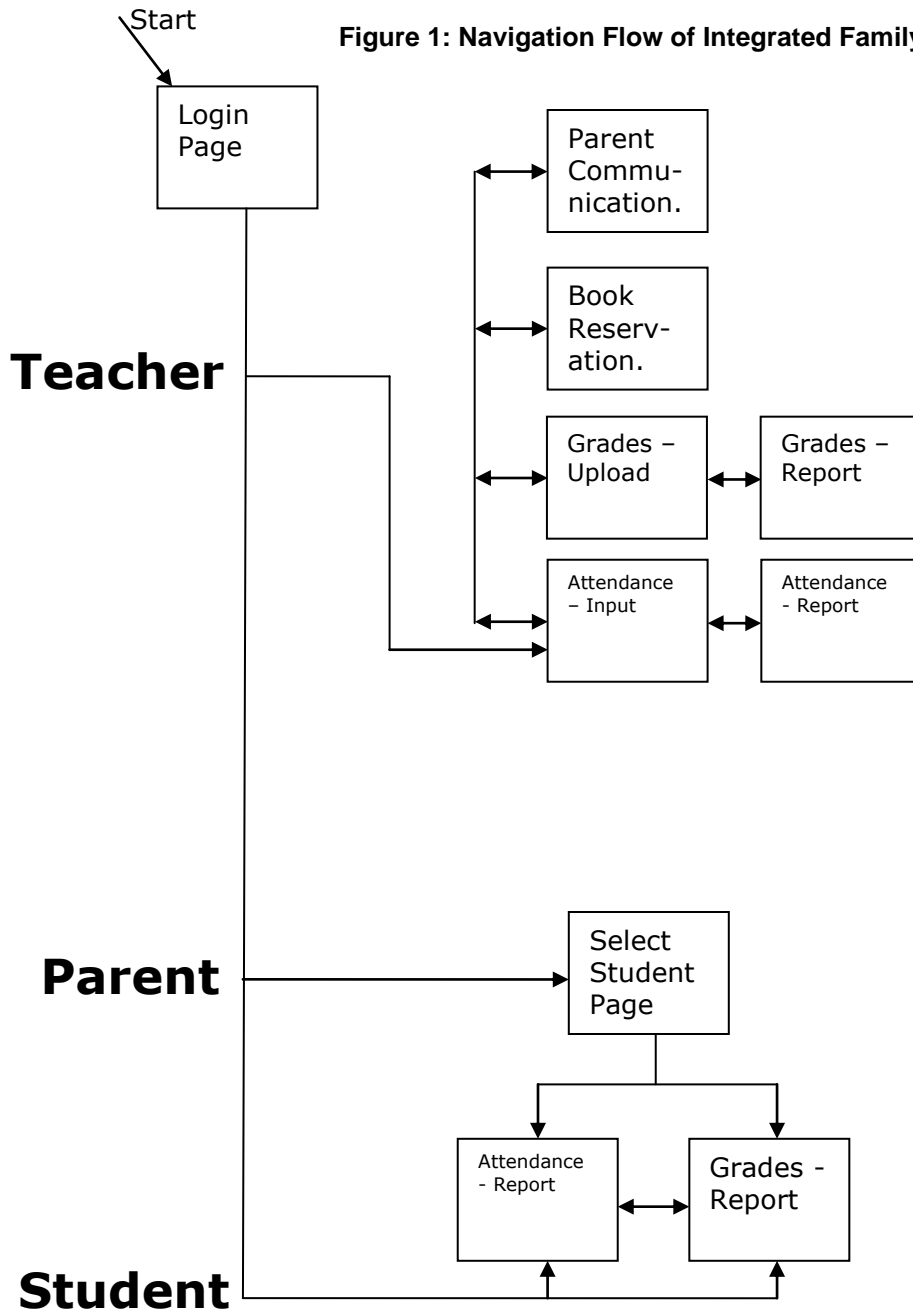
1.2 Status of the prototype

This document is part of the DC package and is the second version. The current phase is the Foundation Phase. A better understanding of the requirements has led to improvements in the UI of the web site as well as the program flow. In addition, prototypes for the text messaging service as well as parsing data from Easy Grade Pro have been added.

2. Navigation Flow

Note: the Parent View is equivalent to the Student View. The client has not identified any information that is different between the two. All pages will have a link to the Logout page, but most of those links are not shown for clarity.

Figure 1: Navigation Flow of Integrated Family Accountability System



3. Critical Issues and Risks

Name	Description
UI Look and Feel	The clients have reviewed the prototype and expressed overall satisfaction with it. There were still a couple requests, however. One such request was to upload student grades through an exported text file rather than manual input into this system.
Report Content	The client has requested a number of specific reports, but they are bound to want more as they see the prototype and continue to consider what would be best for their students.
Cross-Project Cooperation	Client wants this system to interface/combine with a system from another project – the class scheduling and registration system (team 12). This will be used to determine a student’s progress toward graduation (credit-wise).
Page Load Time	One of the current systems takes several minutes to load. Client wants this time to be cut to 5 – 10 seconds. However, it is unclear what the cause of the lag is and they may require a new server to host their own database to speed up the load time.
Uploading Data Exported from Easy Grade Pro	The client has requested that they be able use the “Export” feature of Easy Grade Pro to feed student grades to this system. The risk lies in the fact that this feature was requested recently and the team did not know the format of the exported files.

The clients assisted with this prototype by explaining that they want the UI to look generally like one of their existing products, Easy Grade Pro. They also mentioned a number of the reports that they would like to be included in the site.

Communication via text messaging has been removed from the list now that a functioning prototype is in place (see Section 4.4).

4. Prototypes

The following sections present either a prototype for one of the risks presented in the previous section, or a reference to another document along with an explanation of why that document reduces the risk.

4.1 UI Look and Feel

The following screenshots depict the main screens that the client has requested. As this is a prototype, it is very possible that some of the features shown here may be added to or removed.

4.1.1 Grade Reports

Description	Displays a table of grades as well as a series of graphs that show a student(s) grades.
Related Capability	OC-3 Performance Report, OC-4 Grade Constituent
Pre-condition	User is logged in and requests the "Grades" tab followed by the "View Reports" link.
Post Condition	Student data is displayed in tables/graphs.

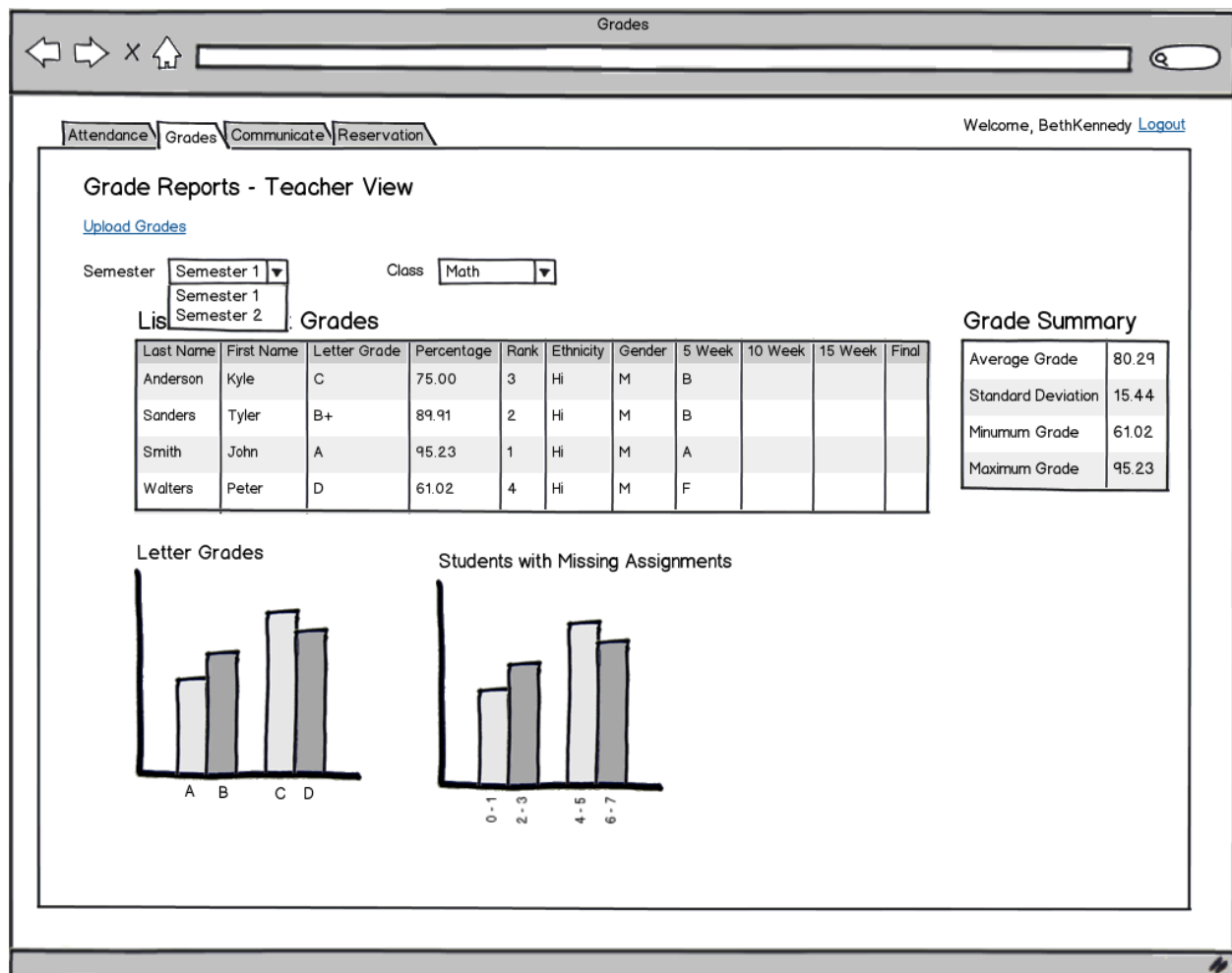


Figure 1.1. Teacher's Grade Report for a Single Class

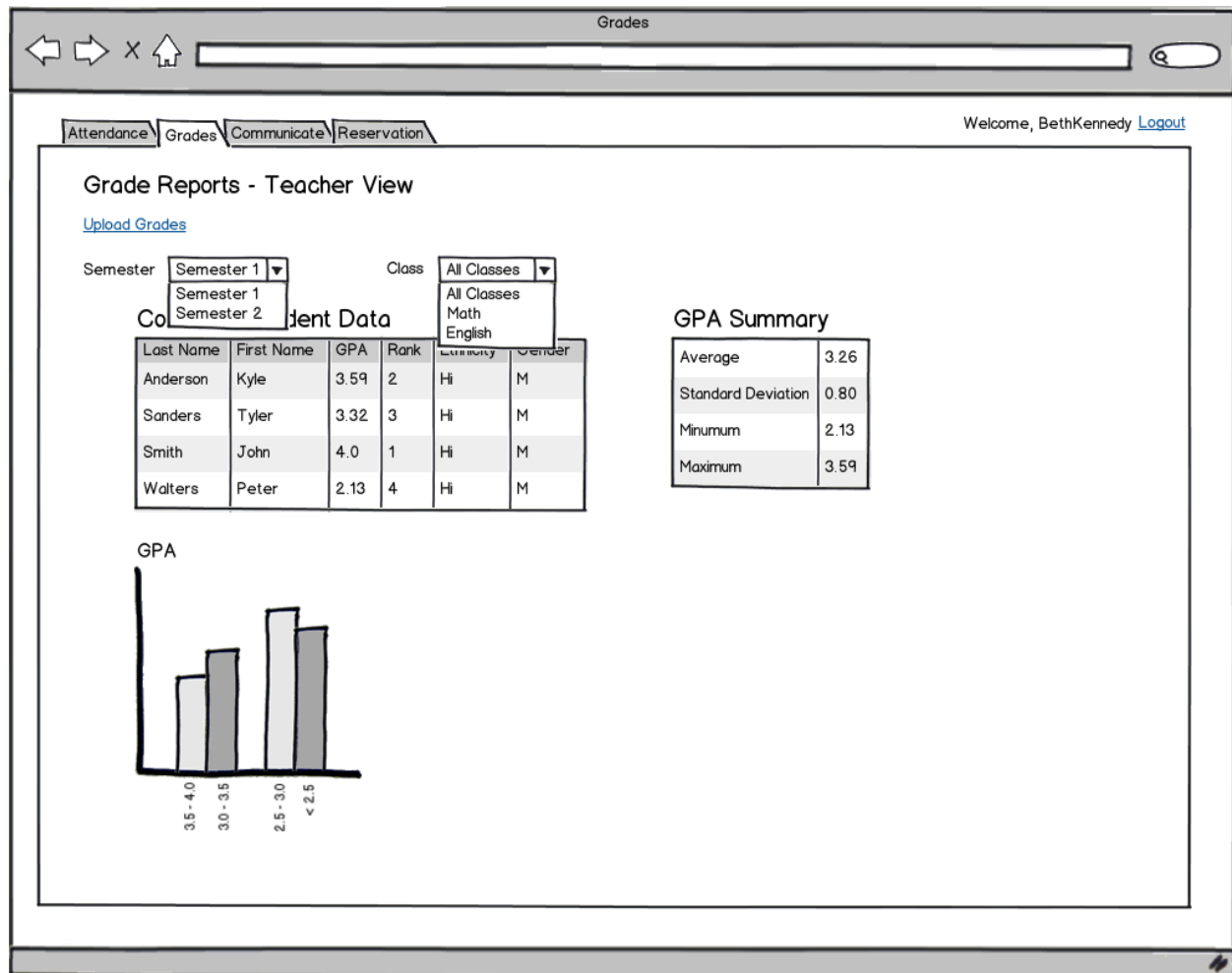


Figure 1.2. Teacher's Grade Report for All Classes

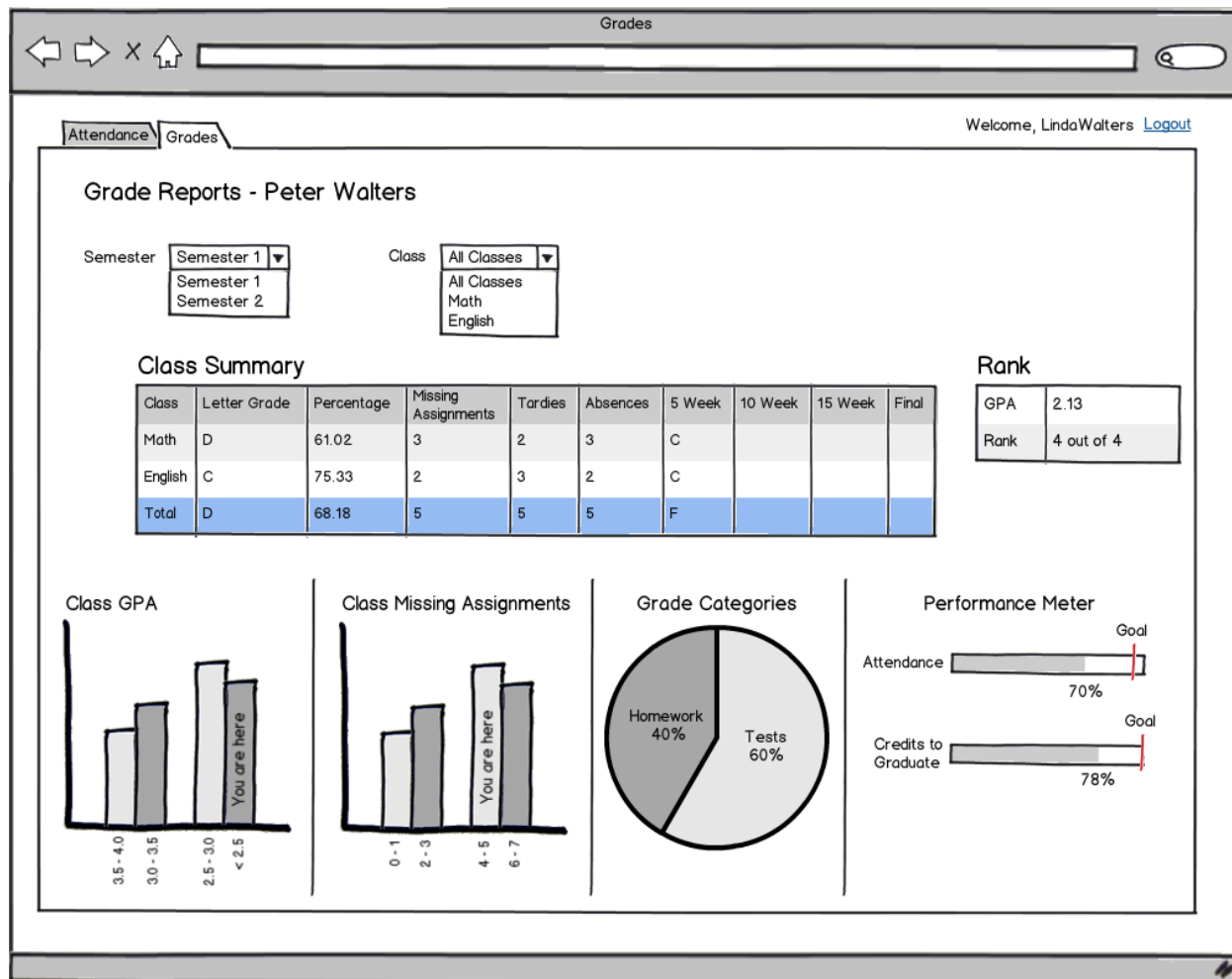


Figure 1.3. Parent's/Student's Grade Report for All Classes

4.1.2 Attendance Reports

Description	Displays a table of grades as well as a series of reports that show a student's attendance across all classes.
Related Capability	OC-3 Performance Report
Pre-condition	User is logged in and requests the "Attendance" tab followed by the "View Reports" link.
Post Condition	Student data is displayed in both tabular and graphical formats.

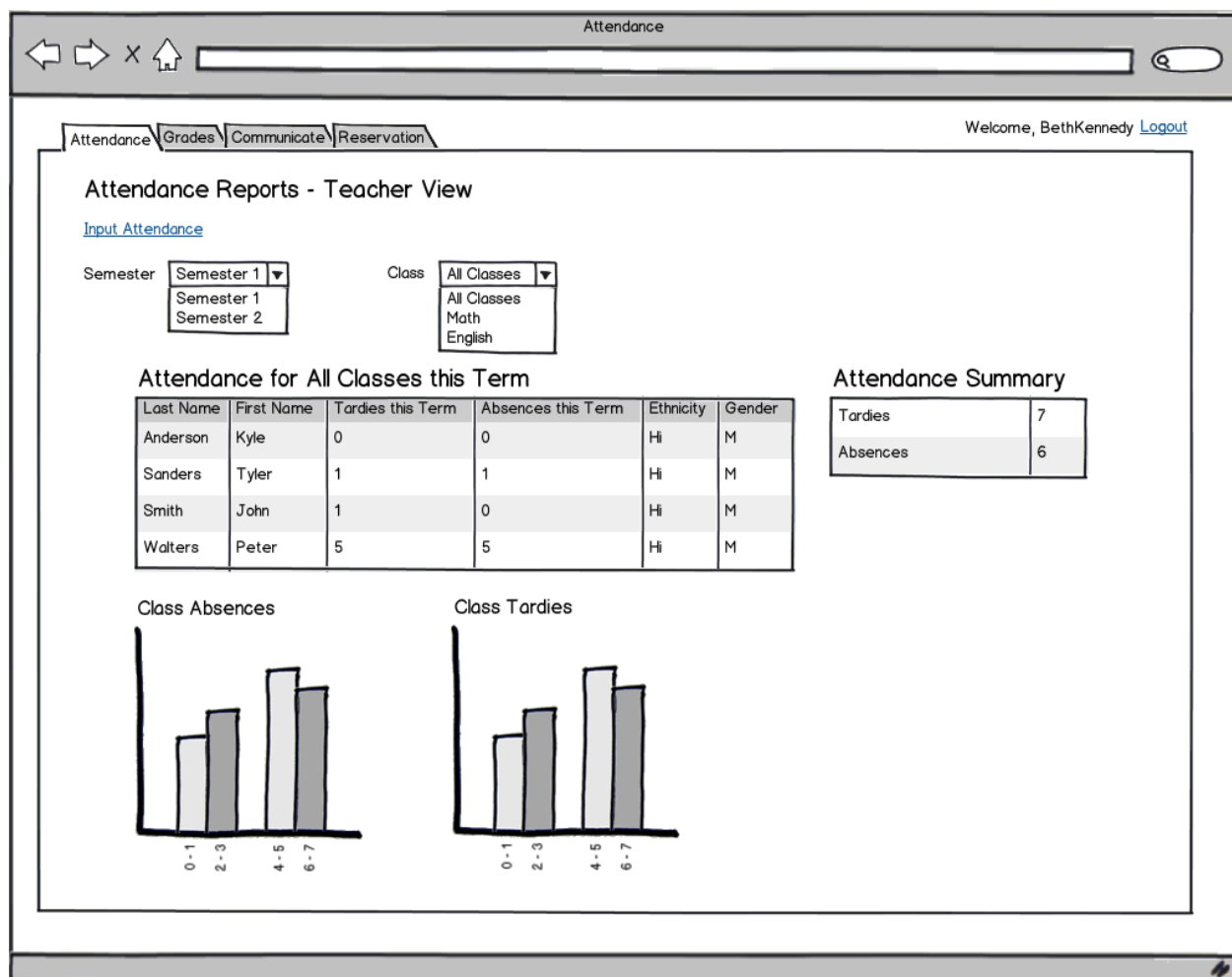


Figure 2.1. Attendance Report for Teachers

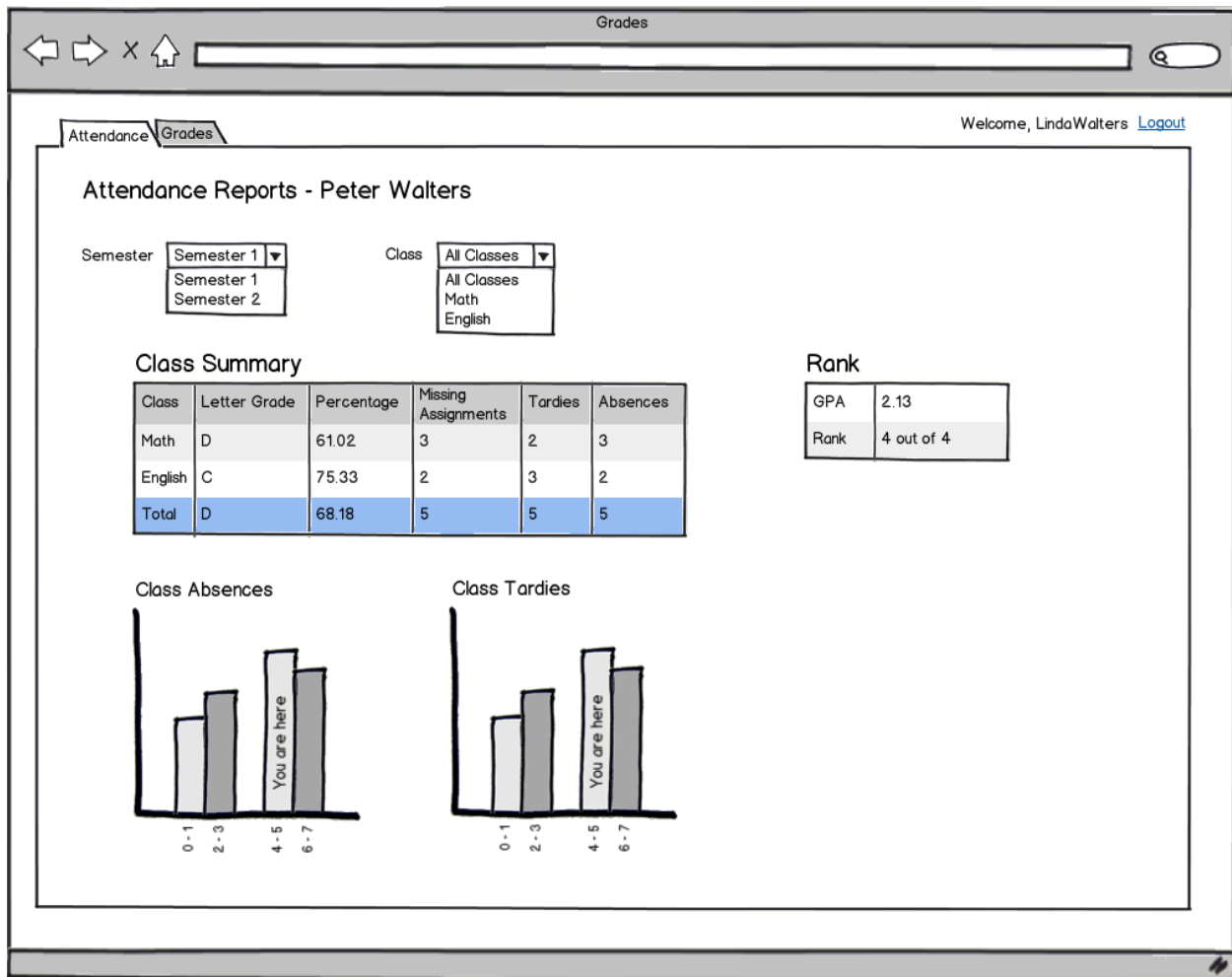


Figure 2.2. Attendance Report for Parents/Students

4.1.3 Grade Upload

Description	Displays an interface that allows the teacher to upload student grades that have been exported by Easy Grade Pro whenever they want.
Related Capability	OC-9 Import from Easy Grade Pro
Pre-condition	User is logged in and requests the "Grades" tab.
Post Condition	Student data is displayed in both tabular and graphical formats.

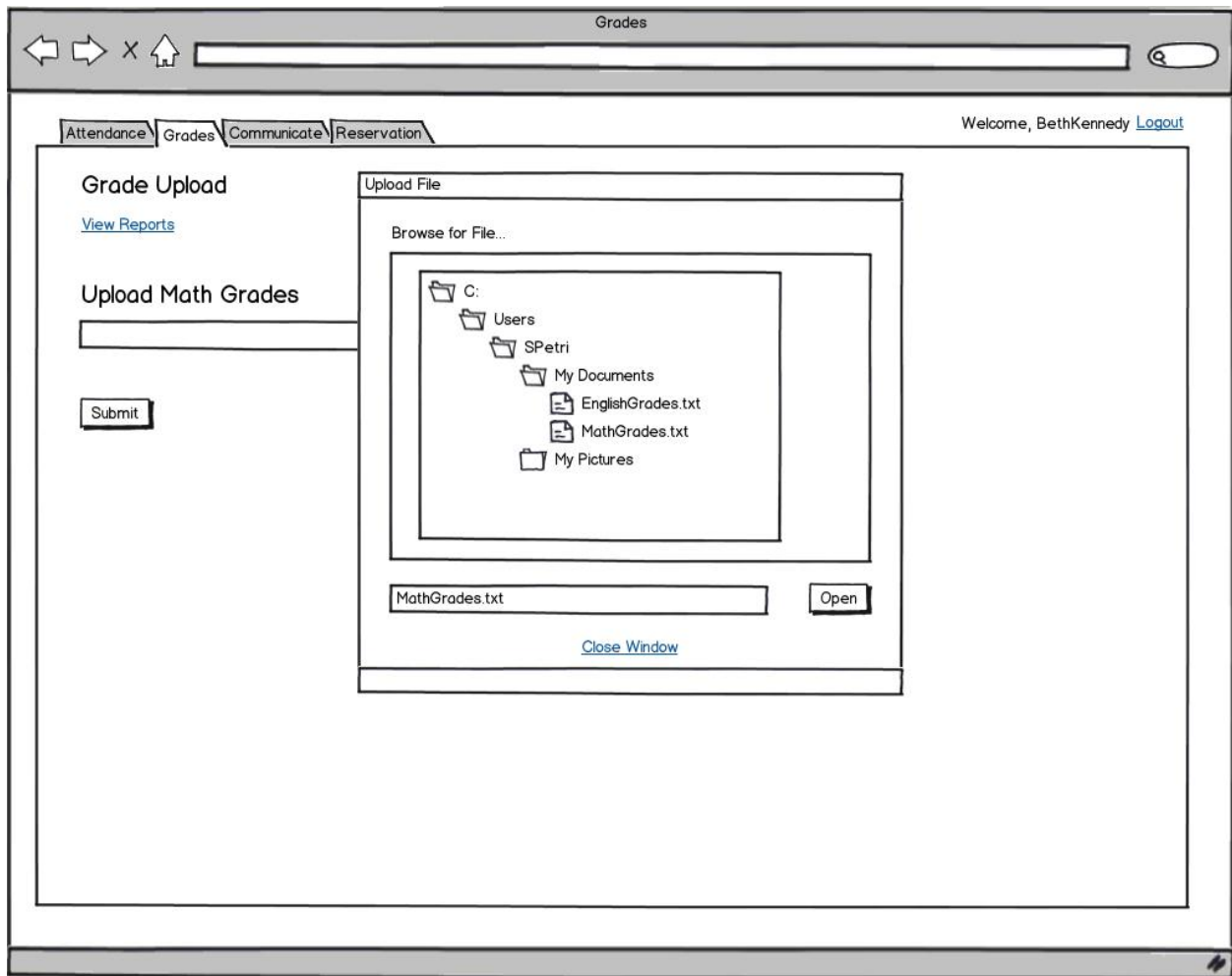


Figure 3. Grade Upload Page

4.1.4 Attendance Input

Description	Displays an interface for each class that the teacher teaches which allows him/her to input student attendance.
Related Capability	OC-3 Performance Report
Pre-condition	User is logged in and requests the "Attendance" tab.
Post Condition	Student data is displayed in both tabular and graphical formats.

Attendance

Welcome, Beth Kennedy [Logout](#)

Attendance Grades Communicate Reservation

Attendance Input

[View Reports](#)

Class: Math

Attendance for Math on 9/28/11

Class Absences

Student Name	Tardies	Absences	Tardy?	Absent?	Email Parent?	Text Parent?	Today, 9/28/11	9/27/11	9/26/11	9/25/11
Anderson, Kyle	0	0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>				
Sanders, Tyler	1	0	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>			T	
Smith, John	0	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>				
Walters, Peter	5	5	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	A		A	

Submit

Figure 4. Attendance Input Page

4.1.5 Communication Page

Description	Displays an interface that allows teachers to send messages to parents, students, and other teachers in the form of emails/texts.
Related Capability	OC-2 Parent Notification
Pre-condition	User is logged in and requests the "Communication" tab.
Post Condition	Teacher is presented with a list of predefined messages to send as either a text or an email.

The screenshot shows a web browser window titled "Notification". Inside, there's a navigation bar with tabs: "Attendance", "Grades", "Communicate", and "Reservation". The "Communicate" tab is active. The main content area is titled "Communicate". It features three dropdown menus: "Send to" (with options: Parent, Teacher, Student), "Class" (with options: Math, <Unspecified>, English), and "Student" (with options: Peter Walters, <Unspecified>, Kyle Anderson, Tyler Sanders, John Smith, Peter Walters). Below these, there's a "Predefined Messages" list with options: "Child was tardy.", "Child was absent.", "Child was given detention.", "Child did not complete assignment." (highlighted), and "Child is failing.". To the right of this list is a "Message Body" text area containing the text "Peter did not complete his Math assignment that was due today.". Further right are "Send by" options: "Text" (radio button) and "Email" (radio button, selected). Below the message body is an "Attach:" field with an "Upload" button. At the bottom left of the main content area is a "Send" button. The top right of the page shows a welcome message "Welcome, Beth Kennedy" and a "Logout" link.

Figure 5. Communication Page

4.1.6 Reservation System

Description	Displays an interface that allows teachers to keep track of materials borrowed by students.
Related Capability	OC-6 Reservation System
Pre-condition	User is logged in and requests the "Reservation" tab.
Post Condition	Teacher is presented with a list of all materials lent out for each of their classes and an easy-to-see indicator of all overdue materials (along with the offending student).

Reservation

Attendance Grades Communicate Reservation

Welcome, Beth Kennedy [Logout](#)

Reservation

Class: Math

Resources Reserved for Math

Last Name	First Name	Book	Date Lent	Date to Return	Email Parent	Text Parent
Anderson	Kyle	Algebra 2	08/29/11	9/25/11	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Sanders	Tyler	Algebra 2	08/29/11	12/11/11	<input type="checkbox"/>	<input type="checkbox"/>
Smith	John	Algebra 2	08/29/11	12/11/11	<input type="checkbox"/>	<input type="checkbox"/>
Walters	Peter	Algebra 2	08/29/11	12/11/11	<input type="checkbox"/>	<input type="checkbox"/>

Submit

This row is being added.

Figure 6. Reservation Page

4.2 Report Content

Refer to the SSRD for a list of all of the reports specified for this system. The clients provided this list and as of this writing, it is a complete list of all reports that they want. Specifically, see:

- CR-2
- CR-5
- CR-7
- CR-13
- CR-18
- CR-19
- CR-21
- CR-22
- CR-24

4.3 Cross-Project Cooperation

The risk presented by working with another team has been mitigated largely by the database design that this team and team 12 have collaborated on. By collaborating early on, we have solidified the schema that we will both use so that this system knows how to access the data stored by team 12. Refer to the SSAD for the database schema.

4.4 Cell Phone Communication Prototype

Description	Uses the API provided by the Mozeo text messaging service to send a text message to a phone.
Related Capability	OC-2 Parent Notification
Pre-condition	User is logged in and requests any page that allows the user to send a text message (such as the Communication page) and has some text messages purchased from Mozeo still remaining. The user submits a phone number and message to send.
Post Condition	The recipient receives a text message.

The clients have requested that they be able to send text messages using an online interface. They recommended that we use Mozeo, which offers many text messages for a flat rate and no monthly fees. Mozeo ended up being a very good choice because they offer an API, 10 free text messages (which we could use for development work), and they provide some code for an text messaging client. Kimberly Krause developed a PHP page that demos the Mozeo service in action. Some of the code is shown below:

```
//variables that came in from the form
$cellNum = $_POST['cellNum'];
$message = $_POST['message'];

echo "You entered cell phone number " . $cellNum . " and message \"" .
$message . "\".<br><br>";

//mozeo API to send the text message.
```

```
$dataArray = array(
    'to'=>$cellNum,
    'datetimestamp'=>time(),
    'messagebody'=>$message,
    'companykey'=>$apiCompanyKey,
    'username'=>$apiUsername,
    'password'=>$apiPassword,
    'stop'=>"yes"
);

//URL encode input parameters
foreach($dataArray as $par=>$value)
{
    $qs=$qs."$par=".urlencode($value)."&";
}
$uri="$apiURL?$qs";

//Send to web service
$cobj=curl_init($uri);
curl_setopt($cobj,CURLOPT_RETURNTRANSFER,1);
$xml=curl_exec($cobj);
curl_close($cobj);
```

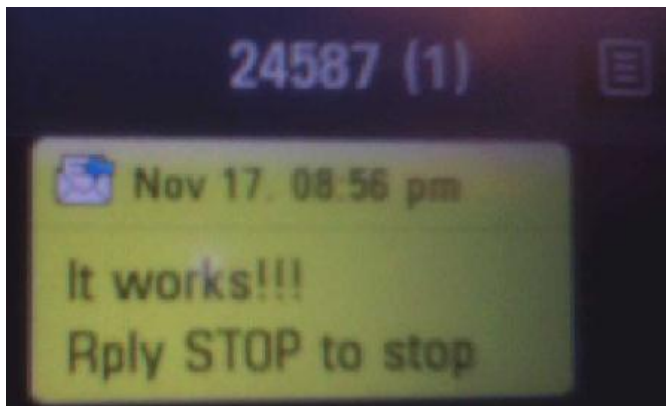
The above code works on top of a UI page whose code is not shown. The page looks as follows:

Cell Phone Prototype

Cell Number: (numbers only)

Message:

Once the "Submit" button is clicked, the page redisplay the message and phone number entered. Shortly afterward, a text message appears:



4.5 Uploading Data Exported from Easy Grade Pro Prototype

Description	Student grade data exported to a text file by Easy Grade Pro is parsed by the system (and eventually uploaded into the database.
Related Capability	OC-9 Import from Easy Grade Pro
Pre-condition	User is logged in and requests the Grade Upload page. User has also already exported a grade file from Easy Grade Pro. User then browses for this file and clicks Upload.
Post Condition	The system parsed the file, picking out useful information, and uploads it to the database.

4.5.1 Report Card Data

Easy Grade Pro has the ability to export data to text files. One such text file contains report card data. The client has requested that letter grades be stored at 5 week intervals for students to view. Easy Grade Pro contains a report card export feature that lists all students and the grades they have received in each of their classes in a text file. The following is a portion of one such test text file:

```

Smith, John      Smith John  020   M    Active      Standard Scale
  English
      Math  1.00
297/300    99.0  A+    1      99.0  A+    297/300    99.0  A+
                        297/300    99.0  A+

297/300    99.0  A+    297/300    99.0  A+

11/19/2011

Johnson, Sally  Johnson    Sally 001   F    Active      Standard
Scale English
      Math  1.00
297/300    99.0  A+    1      99.0  A+    297/300    99.0  A+
                        297/300    99.0  A+

297/300    99.0  A+    297/300    99.0  A+

```

11/19/2011

This data has been organized into tab-delimited columns as specified in the Easy Grade Pro manual. The important columns (along with their numbers) are specified below:

1. Listed name
2. Last name
3. First name
4. Student ID
5. Gender
21. Class Name
22. Class Weight
31. Overall Points
32. Overall Percent
33. Overall Grade
34. Class Rank

There are many other metrics (some not shown in the text above), but these are the important metrics that the client has chosen to record. In order to handle this information, a simple parser was written to separate all the data. The following is a snippet of code from that parser:

```
while(scanner.hasNextLine())
{
    tabCounter = 0;
    String line = scanner.nextLine();
    HashMap<String, String> map = new HashMap<String, String>();
    while(tabCounter < 127)
    {
        switch(tabCounter)
        {
            case 0:
                map.put("Listed Name", line.substring(0,
                    line.indexOf("\t")));
                break;
            case 1:
                map.put("Last Name", line.substring(0,
                    line.indexOf("\t")));
                break;
            case 2:
                map.put("First Name", line.substring(0,
                    line.indexOf("\t")));
                break;
            case 3:
                map.put("ID", line.substring(0,
                    line.indexOf("\t")));
                break;
            case 4:
                map.put("Gender", line.substring(0,
                    line.indexOf("\t")));
                break;
            case 20:
                map.put("Class Name", line.substring(0,
                    line.indexOf("\t")));
                break;
            case 21:
                map.put("Class Weight", line.substring(0,
                    line.indexOf("\t")));
                break;
            case 30:
                map.put("Points", line.substring(0,
                    line.indexOf("\t")));
                break;
        }
    }
}
```

```

        break;
    case 31:
        map.put("Percent", line.substring(0,
            line.indexOf("\t")));
        break;
    case 32:
        map.put("Grade", line.substring(0,
            line.indexOf("\t")));
        break;
    case 33:
        map.put("Rank", line.substring(0,
            line.indexOf("\t")));
        break;
    default:
        break;
    }
    line = line.substring(line.indexOf("\t") + 1);
    tabCounter++;
}

list.add(map);
}

for(int i = 0; i < list.size(); i++)
{
    HashMap<String, String> map = list.get(i);
    System.out.println(map.toString());
    System.out.println();
}

```

The above code creates a scanner that grabs each line from the report card exported text file. Each line represents a single student in a single grade, so each student will have multiple lines in this file. The code then creates a temporary map for the current line in which it will store the 11 metrics listed. Easy Grade Pro's manual says there are 127 metrics, so the code loops through all of them, only pausing at the important ones. Each of those is inserted into the map along with its description. At the very end, all of the important metrics are printed to the screen, which looks like:

```

{Rank=1,
Grade=A+,
Gender=M,
ID=020,
Percent=99.0,
Class Weight=1.00,
Class Name=Math,
Points=297/300,
Last Name=Smith,
Listed Name=Smith, John,
First Name=John}

{Rank=1,
Grade=A+,
Gender=F,
ID=001,
Percent=99.0,
Class Weight=1.00,
Class Name=Math,
Points=297/300,
Last Name=Johnson,
Listed Name=Johnson, Sally,
First Name=Sally}

```

Etc. Note that the order is random due to the nature of a HashMap.

4.5.2 Gradebook Data

Easy Grade Pro also has the ability to export the entire gradebook into an XML formatted text file. This file contains the grades for every assignment for every student in every class. The clients indicated that they intend to upload this detailed information once a week. The truncated XML format is as follows:

```
<class>
  <classrecord>
    <cr_classsubjectname>Math</cr_classsubjectname>
  </classrecord>
  <assignments>
    <assignment>
      <ass_id>1</ass_id>
      <ass_name>HW 1</ass_name>
      <ass_maxscore>100</ass_maxscore>
    </assignment>
  </assignments>
  <student>
    <stud_recordinfo>
      <stud_id>020</stud_id>
      <stud_displayname>Smith, John</stud_displayname>
    </stud_recordinfo>
    <stud_grades>
      <score assid="1">
        <score_raw>99/100</score_raw>
        <score_percent>99.0</score_percent>
        <score_grade>A+ </score_grade>
      </score>
    </stud_grades>
  </student>
</class>
<class> ...
```

Each class contains generic information, such as the subject name, followed by a list of assignments and finally a list of students. Each student in the list has a name and ID (among other pieces of information) and a list of grades for each assignment.

A SAX XML parser was used to parse this document. The three main code segments are the ones that handle the beginning of an element, the contents of an element, and the end of an element. The parser written for this prototype ignores the "assignments" element. Ignoring that element does not increase the risk because "assignments" element is simply another XML element that can be parsed just like those in the code below. The setup/algorithm is as follows:

1. Create a class for each element that contains sub-elements in the XML file. They are Class, ClassRecord, Student, RecordInfo, Grade, and Score. A Student contains both

- a RecordInfo and a list of Grade objects, a Class contains a ClassRecord and a list of Student object, etc.
2. In the main code file, declare one of each of the classes listed above as a global variable.
 3. Define the three required parsing functions – startElement(), characters(), and endElement().
 - a. startElement() – called whenever the parser reads a new element (<example>). The element name as well as all of its attributes are passed in as parameters. This function instantiates a new element based off of the element name. For example, if the element name is "classrecord," a new instance of ClassRecord is created using the global variable from #2. The only other thing to do here is to save the name of the current element name so that the characters() function can reference it.
 - b. Characters() – called whenever the parser reads in the value of an element (<example>value</example>). The contents of the element are passed in as a parameter in the form of a char array. This function is used to populate member variables of classes instantiated from the startElement() function. For example, if the parser sees "A+" inside of the <score_grade> element (it knows which element it is in only because the current element name was stored in the startElement() function), then it will assign this as the letter grade of the Score class (see the XML posted above).
 - c. endElement() – called whenever the parser reads in the end tag of an element (</example>). Since this signals the end of an element, then all of the member variables of its corresponding class must have been filled in. The only action to take here is to save this class to its parent class. For example, when </score> is read, the current <score> element is over and it is safe to save the Score global variable to its parent (which is Grade).
 4. Tell the parser to begin and let it run.

The following is the code of the three functions discussed above:

```
public void startElement(String uri, String localName, String qName,
                        Attributes atts) throws SAXException {
    if(localName.equalsIgnoreCase("class"))
    {
        theClass = new Class();
    }
    else if(localName.equalsIgnoreCase("classrecord"))
    {
        classRecord = new ClassRecord();
    }
    else if(localName.equalsIgnoreCase("student"))
    {
        student = new Student();
    }
    else if(localName.equalsIgnoreCase("stud_recordinfo"))
    {
        recordInfo = new RecordInfo();
    }
    else if(localName.equalsIgnoreCase("stud_grades"))
    {
        grade = new Grade();
    }
    else if(localName.equalsIgnoreCase("score"))
    {

```

```

        {
            score = new Score();
            score.assignmentId = atts.getValue("assid");
        }
        else if(localName.equalsIgnoreCase("overall"))
        {
            overall = new Overall();
        }

        currentName = localName;
        level++;
    }

    public void characters(char[] ch, int start, int length)
        throws SAXException {
        String str = new String(ch);
        str = str.substring(start, start + length);
        if(currentName.equalsIgnoreCase("cr_classsubjectname"))
        {
            classRecord.name += str;
        }
        else if(currentName.equalsIgnoreCase("stud_id"))
        {
            recordInfo.id += str;
        }
        else if(currentName.equalsIgnoreCase("stud_displayName"))
        {
            recordInfo.name += str;
        }
        else if(currentName.equalsIgnoreCase("score_raw"))
        {
            score.score += str;
        }
        else if(currentName.equalsIgnoreCase("score_percent"))
        {
            score.percent += str;
        }
        else if(currentName.equalsIgnoreCase("score_grade"))
        {
            score.grade += str;
        }
        else if(currentName.equalsIgnoreCase("overall_percent"))
        {
            overall.percent += str;
        }
        else if(currentName.equalsIgnoreCase("overall_points"))
        {
            overall.points += str;
        }
        else if(currentName.equalsIgnoreCase("overall_grade"))
        {
            overall.grade += str;
        }
    }

    public void endElement(String uri, String localName, String qName)
        throws SAXException

```



```

{
    level--;

    if(localName.equalsIgnoreCase("class"))
    {
        classes.add(theClass);
    }
    else if(localName.equalsIgnoreCase("classrecord"))
    {
        theClass.classRecord = classRecord;
    }
    else if(localName.equalsIgnoreCase("student"))
    {
        theClass.students.add(student);
    }
    else if(localName.equalsIgnoreCase("stud_recordinfo"))
    {
        student.recordInfo = recordInfo;
    }
    else if(localName.equalsIgnoreCase("stud_grades"))
    {
        student.grades = grade;
    }
    else if(localName.equalsIgnoreCase("score"))
    {
        grade.scores.add(score);
    }
    else if(localName.equalsIgnoreCase("overall"))
    {
        grade.overall = overall;
    }
}

```

Once all of this data has been parsed, it can be displayed in any combination. For example, one can loop through the data to show the assignments per student in each class by using the following code:

```

for(int i = 0; i < handler.classes.size(); i++)
{
    System.out.println();
    System.out.println(handler.classes.get(i).classRecord.name);
    System.out.println("-----");
    for(int j = 0; j < handler.classes.get(i).students.size(); j++)
    {
        Student student = handler.classes.get(i).students.get(j);
        System.out.println(student.recordInfo.name + " (" +
            student.recordInfo.id + ")");
        for(int k = 0; k < student.grades.scores.size(); k++)
        {
            Score score = student.grades.scores.get(k);
            System.out.println("Assignment " +
                score.assignmentId);
            System.out.println(score.score);
            System.out.println(score.percent);
            System.out.println(score.grade);
            if(k < student.grades.scores.size() - 1)

```

```

        {
            System.out.println();
        }
    }
    if(j < handler.classes.get(i).students.size() - 1)
    {
        System.out.println("-----");
    }
}
}

```

And the output looks like:

```

Math
-----
Smith, John (020)
Assignment 1
99/100
99.0
A+
Assignment 2
100/100
100.0
A+
Assignment 3
98/100
98.0
A+
-----
Johnson, Sally (001)
Assignment 1
100/100
100.0
A+
...

```

The above is real output produced after looping through the parsed data.

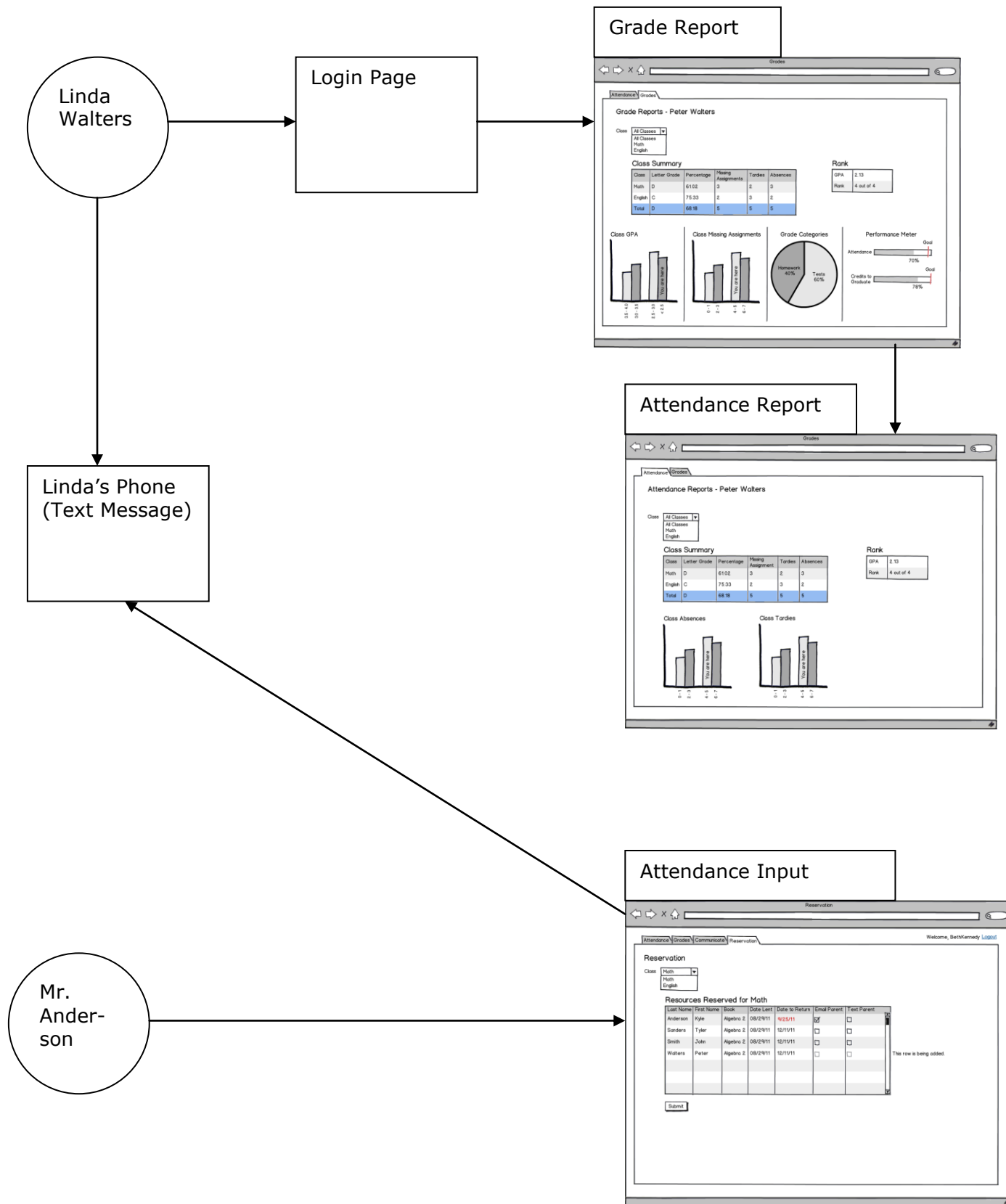
5. Use Cases

5.1 Simple Use Case

Mr. Anderson, a math teacher, takes the first attendance of the day by logging into the LEMA Integrated Family Accountability System and navigating to the attendance input screen (Figure 4). He notices that John Smith is tardy and that Peter Walters is absent and checks the corresponding boxes. He can see from the table Peter has a bad attendance record and decides to send an auto-generated text message to his parents by clicking the corresponding check box. Mr. Anderson can also see that John has had a perfect attendance record until today and decides to make sure that John's parents know that he is tardy by checking the "Email" box. Mr. Anderson clicks the Submit button and continues with class...

5.2 Extended Case with Walkthrough

Linda Walters (Peter Walters' mother) mother decides to check on her son's performance in school, so she logs into the LEMA Integrated Family Accountability System and navigates to the grade reports. Since she is a parent, she is directed to the parent/student view, which contains more charts than the teacher view for fast, easy comprehension (Figure 1.3). She sees that Peter is not doing well and that his grades are not good compared to many of the other children at LEMA. She then checks the attendance report by clicking the attendance tab and once again is directed to the parent/student view (Figure 2.2). She notices that Peter has been absent 5 times this term. She then receives a text message from one of Peter's teachers (Mr. Anderson) about Peter's absence today...



5.3 Uploading Student Grades Use Case

Mr. Jones hears the final bell of the day ring and dismisses his Math class students. Realizing that today marks the end of the fifth week of classes, he goes to his computer and opens his grading program, Easy Grade Pro. He then navigates to File->Export, chooses the report card format, and exports the data to a text file. He logs into the Family Accountability System and navigates to the Grade Upload page, clicks on the Browse button, finds the file he just exported, and uploads it. He then checks Peter Walters' grade page of Peter Walters and notices that there is a column that contains his 5-week letter grade.

5.4 Uploading Assignments Use Case

It is a Friday evening and Mr. Clark has just finished grading the assignments of the week. Within Easy Grade Pro, he chooses File->Export, chooses the Gradebook format, and exports the data to an XML text file. He then logs in to the Family Accountability System and navigates to the Grade Upload page. He clicks the Browse button, finds the file he just exported, and uploads it. He then checks Peter Walters' grade page and sees that all of the most recent assignments are accounted for.